Learning Deep Time-index Models for Time Series Forecasting

Gerald Woo [1 2] **Chenghao Liu** [1] **Doyen Sahoo** [1] **Akshat Kumar** [2] **Steven Hoi** [1]

Abstract

Deep learning has been actively applied to time series forecasting, leading to a deluge of new methods, belonging to the class of *historical-value* models. Yet, despite the attractive properties of *time-index* models, such as being able to model the continuous nature of underlying time series dynamics, little attention has been given to them. Indeed, while naive deep time-index models are far more expressive than the manually predefined function representations of classical time-index models, they are inadequate for forecasting, being unable to generalize to unseen time steps due to the lack of inductive bias. In this paper, we propose DeepTime, a meta-optimization framework to learn deep time-index models which overcome these limitations, yielding an efficient and accurate forecasting model. Extensive experiments on real world datasets in the long sequence time-series forecasting setting demonstrate that our approach achieves competitive results with state-of-the-art methods, and is highly efficient. Code is available at `https://github.com/salesforce/DeepTime`.

1. Introduction

Time series forecasting has important applications across business and scientific domains, such as demand forecasting (Carbonneau et al., 2008), capacity planning and management (Kim, 2003), and anomaly detection (Laptev et al., 2017). There are two typical approaches to time series forecasting – historical-value, and time-index models. Historical-value models predict future time step(s) as a function of past observations, $\hat{\boldsymbol{y}}_{t+1} = f(\boldsymbol{y}_t, \boldsymbol{y}_{t-1}, \ldots)$ (Benidis et al., 2020), while time-index models, a less studied approach, are defined to be models whose predictions are

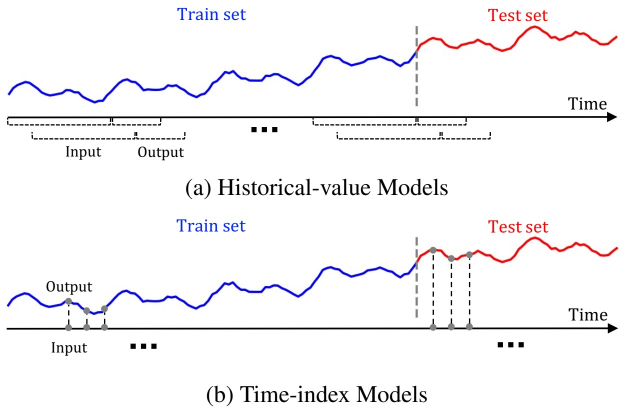


Figure 1. Visual comparison between the two paradigms of historical-value, and time-index models. After the models are trained on the training set, forecasts are made over the test set, which may be arbitrarily long. Historical-value models make predictions conditioning on an input lookback window, whereas time-index models do not make use of new incoming information during the test phase.

purely functions of the corresponding time-index features at future time-step(s), $\hat{\boldsymbol{y}}_{t+1} = f(\boldsymbol{\tau}_{t+1})$ (see Figure 1 for a visual comparison). Historical-value models have been widely used due to their simplicity. However, they can only model temporal relationships at the data sampling frequency. This is an issue since time series observations available to us tend to have a much lower resolution (sampled at a lower frequency) than the underlying dynamics (Gong et al., 2015; 2017), leading historical-value models to be prone to capturing spurious correlations. On the other hand, time-index models intrinsically avoid this problem, directly modeling the mapping from time-index features to predictions in continuous space and learning signal representations which change smoothly and correlate with each other in continuous space.

Classical time-index models (Taylor & Letham, 2018; Hyndman & Athanasopoulos, 2018; Ord et al., 2017) rely on predefined function forms to generate predictions. They optionally follow the structural time-series formulation (Harvey & Shephard, 1993), $\boldsymbol{y}_t = g(\boldsymbol{\tau}_t) + s(\boldsymbol{\tau}_t) + h(\boldsymbol{\tau}_t)$, where g, s, h represent trend, periodic, and holiday components respectively. For example, g could be predefined as a linear, polynomial, or even a piece-wise linear function. While these functions are simple and easy to learn, they

[1]Salesforce Research Asia [2]School of Computing and Information Systems, Singapore Management University. Correspondence to: Gerald Woo <gwoo@salesforce.com>, Chenghao Liu <chenghao.liu@salesforce.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

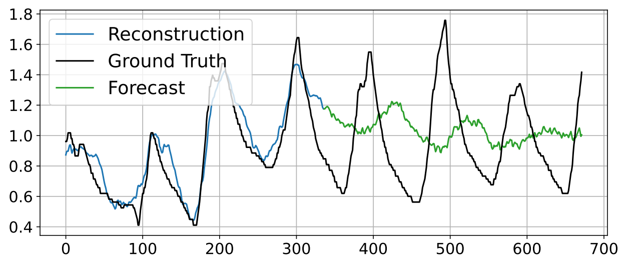


Figure 2. Ground truth time series and predictions from a vanilla deep time-index model. The reconstruction of historical training data as well as forecast over the horizon is visualized. Observe that it overfits to historical data and is unable to extrapolate. This model corresponds to +Local in Table 4 of our ablations.

have limited capacity, are unable to fit more complex time series, and such predefined function forms may be too stringent of an assumption which may not hold in practice. While it is possible to perform model selection across various function forms, this requires either strong domain expertise, or computationally heavy cross-validation across a large set of parameters.

Deep learning gives a seemingly natural solution to this problem faced by classical time-index models – parameterize $f(\boldsymbol{\tau}_t)$ as a neural network, and learn the function form in a *purely* data-driven manner. While neural networks are extremely expressive with a strong capability to approximate complex functions, we argue that when trained via standard supervised learning, they face the debilitating problem of an inability to extrapolate across the forecast horizon, i.e. unable to generalize to future time step(s). This is visualized in Figure 2, where a deep time-index model is able to fit the training data well in the interval $[0, T']$, but it is unable to produce meaningful forecasts for the interval $[T', T]$. This arises due to the lack of extrapolation capability of neural networks when used in time-index models. While classical time-index models have limited expressitivity, the strong inductive biases (e.g. linear trend, periodicity) instructs their predictions over unseen time steps. On the other hand, vanilla deep time-index models do not have such inductive biases, and thus, while they do learn an appropriate function form over the training data, achieving extremely good reconstruction loss, they have little to no generalization capabilities over unseen time steps.

We now raise the question – how do we achieve the best of both worlds – to retain the flexibility and expressiveness of the neural network, as well as to learn inductive bias to guarantee extrapolation capability over unseen time steps from time series data? We present a solution, DeepTime, a meta-optimization framework to learn deep time-index models for time series forecasting. Our framework splits the learning process of deep time-index models into two stages, the inner, and outer learning process. The inner learning process acts as the standard supervised learning process, fitting parameters to recent time steps. The outer learning process enables the deep time-index model to learn a strong inductive bias for extrapolation from data. We summarize our contributions as follows:

- We introduce the use of deep time-index models for time series forecasting, proposing a meta-optimization framework to address their shortcomings, making them viable for forecasting.
- We introduce a specific function form for deep time-index models by leveraging implicit neural representations (Sitzmann et al., 2020b), and a novel concatenated Fourier features module to efficiently learn high frequency patterns in time series. We also specify an efficient instantiation of the meta-optimization procedure via a closed-form ridge regressor (Bertinetto et al., 2019).
- We conduct extensive experiments on the long sequence time series forecasting (LSTF) setting, demonstrating DeepTime to be extremely competitive with state-of-the-art baselines. At the same time, DeepTime is highly efficient in terms of runtime and memory.

2. Related Work

Neural Forecasting Neural forecasting (Benidis et al., 2020) methods have seen great success in recent times. One related line of research are Transformer-based methods for LSTF (Li et al., 2019; Zhou et al., 2021; Xu et al., 2021; Woo et al., 2022; Zhou et al., 2022) which aim to not only achieve high accuracy, but to overcome the vanilla attention's quadratic complexity. Fully connected methods (Oreshkin et al., 2020; Challu et al., 2022) have also shown success, with (Challu et al., 2022) introducing hierarchical interpolation and multi-rate data sampling for the LSTF task. Bi-level optimization in the form of meta-learning and the use of differentiable closed-form solvers has been explored in time series forecasting (Grazzi et al., 2021), for the purpose of adapting to new time series datasets, where tasks are defined to be the entire time series.

Time-index Models Time-index models take as input time-index features such as datetime features to predict the value of the time series at that time step. They have been well explored as a special case of regression analysis (Hyndman & Athanasopoulos, 2018; Ord et al., 2017), and many different predictors have been proposed for the classical setting,including linear, polynomial, and piecewise linear trends, and dummy variables indicating holidays. Of note, Fourier terms have been used to model periodicity, or seasonal patterns, and is also known as harmonic regression (Young et al., 1999). Prophet (Taylor & Letham, 2018) is a popular classical approach which uses a structural time series formulation, specialized for business forecasting. Another classical approach of note are Gaussian Processes (Rasmussen, 2003; Corani et al., 2021) which are non-parametric models, often requiring complex kernel engineering. (Godfrey & Gashler, 2017) introduced an initial attempt at using time-index based neural networks to fit a time series for forecasting. Yet, their work is more reminiscent of classical methods, manually specifying periodic and non-periodic activation functions, analogous to the representation functions.

Implicit Neural Representations INRs have recently gained popularity in the area of neural rendering (Tewari et al., 2021). They parameterize a signal as a continuous function, mapping a coordinate to the value at that coordinate. A key finding was that positional encodings (Mildenhall et al., 2020; Tancik et al., 2020) are critical for ReLU MLPs to learn high frequency details, while another line of work introduced periodic activations (Sitzmann et al., 2020b). Meta-learning on via INRs have been explored for various data modalities, typically over images or for neural rendering tasks (Sitzmann et al., 2020a; Tancik et al., 2021; Dupont et al., 2021), using both hypernetworks and optimization-based approaches. (Yüce et al., 2021) show that meta-learning on INRs is analogous to dictionary learning. In time series, (Jeong & Shin, 2022) explored using INRs for anomaly detection, opting to make use of periodic activations and temporal positional encodings.

3. DeepTime

In this section, we first formally describe the time series forecasting problem, and how to use time-index models for this problem setting. Next, we introduce the model architecture specifics for deep time-index models. Finally, we introduce the generic form of our meta-optimization framework, then specifically, how to use of a differentiable closed-form ridge regression module to perform the meta-optimization efficiently. Pseudocode of DeepTime is available in Appendix A.

Problem Formulation In time series forecasting, we consider a time series dataset $(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$, where $\boldsymbol{y}_t \in \mathbb{R}^m$ is the m-dimension observation at time t. Consider a train-test split such that the range $(1, \ldots, T')$ is considered to be the training set and the range $(T'+1, \ldots, T)$ is the test set, where $T - T' \geq H$. The goal is to construct point forecasts over a horizon of length H, over the test set, $\hat{\boldsymbol{Y}}_{t:t+H} = [\hat{\boldsymbol{y}}_t; \ldots; \hat{\boldsymbol{y}}_{t+H-1}], \forall t = T'+1, T'+2, \ldots, T'-H+1$. A time-index model, $f : \mathbb{R} \to \mathbb{R}^m, f : \boldsymbol{\tau}_t \mapsto \hat{\boldsymbol{y}}_t$, achieves this by minimizing a reconstruction loss $\mathcal{L} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ over the training set, where $\boldsymbol{\tau}_t$ is a time-index feature for which values are known for all time steps. Then, we can query it over the test set to obtain forecasts, $\hat{\boldsymbol{Y}}_{t:t+H} = f(\boldsymbol{\tau}_{t:t+H})$.

3.1. Deep Time-index Model Architecture

INRs (Sitzmann et al., 2020b) are a class of coordinate-based models, mapping coordinates to values, which have been extensively studied. Time-index models are a case of 1d coordinate-based models, thus, we leverage this existing class of models, which are essentially a stack of multi-layered perceptrons as our proposed deep time-index model architecture. Visualized in Figure 3a, a K-layered, ReLU (Nair & Hinton, 2010) INR is a function $f_\theta : \mathbb{R}^c \to \mathbb{R}^m$ where:

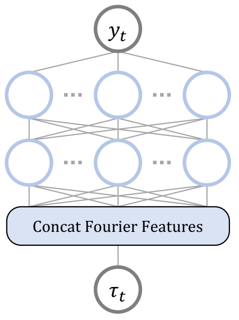
$$\begin{aligned} \boldsymbol{z}^{(0)} &= \boldsymbol{\tau} \\ \boldsymbol{z}^{(k+1)} &= \max(0, \boldsymbol{W}^{(k)}\boldsymbol{z}^{(k)} + \boldsymbol{b}^{(k)}), \quad k = 0, \ldots, K-1 \\ f_\theta(\boldsymbol{\tau}) &= \boldsymbol{W}^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)} \end{aligned} \tag{1}$$

where $\boldsymbol{\tau} \in \mathbb{R}^c$ are time-index features. MLPs have shown to experience difficulty in learning high frequency functions, this problem known as "spectrial bias" (Rahaman et al., 2019; Basri et al., 2020). Coordinate-based methods suffer from this issue in particular when trying to represent high frequency content present in the signal. Tancik et al. (2020) introduced a random Fourier features layer which allows INRs to fit to high frequency functions, by modifying $\boldsymbol{z}^{(0)} = \gamma(\boldsymbol{\tau}) = [\sin(2\pi\boldsymbol{B}\boldsymbol{\tau}), \cos(2\pi\boldsymbol{B}\boldsymbol{\tau})]^T$, where each entry in $\boldsymbol{B} \in \mathbb{R}^{d/2 \times c}$ is sampled from $\mathcal{N}(0, \sigma^2)$ with d is the hidden dimension size of the INR and σ^2 is the scale hyperparameter. $[\cdot, \cdot]$ is a row-wise stacking operation.

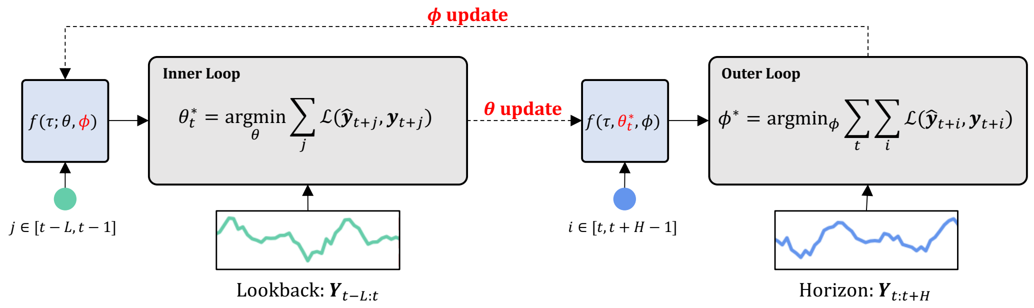
Concatenated Fourier Features While the random Fourier features layer endows INRs with the ability to learn high frequency patterns, one major drawback is the need to perform a hyperparameter sweep for each task and dataset to avoid over or underfitting. We overcome this limitation with a simple scheme of concatenating multiple Fourier basis functions with diverse scale parameters, i.e. $\gamma(\boldsymbol{\tau}) = [\sin(2\pi\boldsymbol{B}_1\boldsymbol{\tau}),\ \cos(2\pi\boldsymbol{B}_1\boldsymbol{\tau}),\ \ldots,\ \sin(2\pi\boldsymbol{B}_S\boldsymbol{\tau}), \cos(2\pi\boldsymbol{B}_S\boldsymbol{\tau})]^T$, where elements in $\boldsymbol{B}_s \in \mathbb{R}^{d/2 \times c}$ are sampled from $\mathcal{N}(0, \sigma_s^2)$, and $\boldsymbol{W}^{(0)} \in \mathbb{R}^{d \times Sd}$. We perform an analysis in Section 5.3 and show that the performance of our proposed concatenated Fourier features (CFF) does not significantly deviate from the setting with the optimal scale parameter obtained from a hyperparameter sweep.

3.2. Meta-optimization Framework

Explained in Section 1, vanilla deep time-index models are unable to perform forecasting due to their failure in extrapolating beyond observed time-indices. Formally, the original hypothesis class of time-index model is denoted $\mathcal{H}_{\text{INR}} = \{f(\tau; \theta) \mid \theta \in \Theta\}$, where Θ is the parameter space. The original hypothesis class is too expressive, providing no guarantees that training on the lookback window leads to good extrapolation. To solve this, our meta-optimization framework learns an inductive bias for the deep time-index



(a) Deep Time-index Model Architecture


ϕ update
$f(\tau;\theta,\phi)$
Inner Loop
$\theta_t^* = \text{argmin}_\theta \sum_j \mathcal{L}(\hat{\boldsymbol{y}}_{t+j}, \boldsymbol{y}_{t+j})$
θ update
$f(\tau,\theta_t^*,\phi)$
Outer Loop
$\phi^* = \text{argmin}_\phi \sum_t \sum_i \mathcal{L}(\hat{\boldsymbol{y}}_{t+i}, \boldsymbol{y}_{t+i})$
$j \in [t-L, t-1]$
$i \in [t, t+H-1]$
Lookback: $\boldsymbol{Y}_{t-L:t}$
Horizon: $\boldsymbol{Y}_{t:t+H}$

(b) Meta-optimization Framework

Figure 3. Left: Our proposed deep time-index model has a simple overall architecture, comprising a concatenated Fourier features, and several layers of MLPs. Right: The meta-optimization framework aims to learn meta parameters of deep time-index models by following a bi-level optimization problem formulation (Equations (2) and (3)). The inner loop is optimized over the lookback window, yielding the optimal base parameters for a locally stationary distribution, θ_t^*. Each lookback window has it's own optimal base parameters, which are then used to predict the immediate following forecast horizon. This composes the outer loop, which optimizes the meta parameters over the entire time series, yielding an inductive bias which enables extrapolation over forecast horizons, given the optimal base parameters.

model from data. Firstly, rather than using the entire training set in a naive supervised learning setting, whereby older training points provide no additional benefit in learning a time-index model for extrapolation, we split the time series into lookback window and forecast horizon pairs. Let the L time steps preceding the forecast horizon at time step t, be the lookback window, $\boldsymbol{Y}_{t-L:t} = [\boldsymbol{y}_{t-L}; \dots; \boldsymbol{y}_{t-1}]^T \in \mathbb{R}^{L\times m}$. Next, consider the case where we split the model parameters into two, possibly overlapping subsets, $\phi \in \Phi$ and $\theta \in \Theta$, known as the meta and base parameters, respectively, where Φ is the parameter space of the meta parameters. The meta parameters are responsible for learning the inductive bias from multiple lookback window and forecast horizon pairs from the training data, while the base parameters aim to learn and adapt quickly to the lookback window at test time. Thus, we aim to encode an inductive bias in ϕ, **learned** to enable extrapolation across the forecast horizon when the base parameters adapt corresponding lookback window, resulting in $\mathcal{H}_{\text{Meta}} = \{f(\tau;\theta,\phi^*) \mid \theta \in \Theta\}$. This is achieved by formulating the bi-level optimization problem:

$$\phi^* = \arg\min_{\phi} \sum_{t=L+1}^{T-H+1} \sum_{i=0}^{H-1} \mathcal{L}(f(\boldsymbol{\tau}_{t+i};\theta_t^*,\phi), \boldsymbol{y}_{t+i}) \tag{2}$$

$$s.t. \quad \theta_t^* = \arg\min_{\theta} \sum_{j=-L}^{-1} \mathcal{L}(f(\boldsymbol{\tau}_{t+j};\theta,\phi), \boldsymbol{y}_{t+j}) \tag{3}$$

Illustrated in Figure 3b, Equation (2) represents the outer loop, and Equation (3), the inner loop. The first summation in the outer loop over index t represents iterating over the time steps of the dataset, and the second summation over index i represents each time step of the forecast horizon. The summation in the inner loop over index j represents each time step of the lookback window.

Fast and Efficient Meta-optimization Following the above generic formulation of the meta-optimization framework to learn deep time-index models, we now describe a specific instantiation of the framework which enables both training and forecasting at test time to be fast and efficient. Similar bi-level optimization problems have been explored (Ravi & Larochelle, 2017; Finn et al., 2017) and one naive approach is to directly backpropagate through inner gradient steps. However, such methods are highly inefficient, have many additional hyperparameters, and are instable during training (Antoniou et al., 2019). Instead, to achieve speed and efficiency, we specify that the base parameters consist of only the last layer of the INR, while the rest of the INR are meta parameters. Thus, the inner loop optimization only applies to this last layer. This transforms the inner loop optimization problem into a simple ridge regression problem for the case of mean squared error loss, having a simple analytic solution to replace the otherwise complicated non-linear optimization problem (Bertinetto et al., 2019).

Formally, for a K-layered model, $\phi = \{\boldsymbol{W}^{(0)}, \boldsymbol{b}^{(0)}, \dots, \boldsymbol{W}^{(K-1)}, \boldsymbol{b}^{(K-1)}, \lambda\}$ are the meta parameters and $\theta = \{\boldsymbol{W}^{(K)}\}$ are the base parameters, following notation from Equation (1). Then let $g_\phi : \mathbb{R} \to \mathbb{R}^d$ be the meta learner where $g_\phi(\boldsymbol{\tau}) = \boldsymbol{z}^{(K)}$. For a lookback-horizon pair, $(\boldsymbol{Y}_{t-L:t}, \boldsymbol{Y}_{t:t+H})$, the features of the lookback window obtained from the meta learner is denoted $\boldsymbol{Z}_{t-L:t} = [g_\phi(\boldsymbol{\tau}_{t-L}); \dots; g_\phi(\boldsymbol{\tau}_{t-1})]^T \in \mathbb{R}^{L\times d}$, where $[\cdot;\cdot]$ is a column-wise concatenation operation. The inner loop thus solves the optimization problem:

$$\begin{aligned} \boldsymbol{W}_t^{(K)*} &= \arg\min_{\boldsymbol{W}} ||\boldsymbol{Z}_{t-L:t}\boldsymbol{W} - \boldsymbol{Y}_{t-L:t}||^2 + \lambda||\boldsymbol{W}||^2 \\ &= (\boldsymbol{Z}_{t-L:t}^T\boldsymbol{Z}_{t-L:t} + \lambda\boldsymbol{I})^{-1}\boldsymbol{Z}_{t-L:t}^T\boldsymbol{Y}_{t-L:t} \end{aligned} \tag{4}$$

Now, let $\boldsymbol{Z}_{t:t+H} = [g_\phi(\boldsymbol{\tau}_t); \dots; g_\phi(\boldsymbol{\tau}_{t+H-1})]^T \in \mathbb{R}^{H\times d}$

be the features of the forecast horizon. Then, our predictions are $\hat{\boldsymbol{Y}}_{t:t+H} = \boldsymbol{Z}_{t:t+H}\boldsymbol{W}_t^{(K)*}$. This closed-form solution is differentiable, which enables gradient updates on the parameters of the meta learner, ϕ. A bias term can be included for the closed-form ridge regressor by appending a scalar 1 to the feature vector $g_\phi(\boldsymbol{\tau})$. The end result of training DeepTime on a dataset is the restricted hypothesis class $\mathcal{H}_{\text{DeepTime}} = \left\{g_{\phi^*}(\boldsymbol{\tau})^T\boldsymbol{W}^{(K)} \mid \boldsymbol{W}^{(K)} \in \mathbb{R}^{d\times m}\right\}$. Finally, we propose to use relative time-index features, $\boldsymbol{\tau}_{t+i} = \frac{i+L}{L+H-1}$ for $i = -L, -L+1, \dots, H-1$, i.e. a $[0,1]$-normalized time-index.

4. Theoretical Analysis

In the following, we derive a generalization bound of DeepTime under the PAC-Bayes framework (Amit & Meir, 2018). Give a long time series, suppose we can split it into n instances (each has a length L lookback window and length H forecast horizon) for training. Then the k-th instance is denoted $\mathcal{S}_k = \{z_{k-L}, \dots, z_k, \dots, z_{k+H-1}\}$, where $z_t = (\boldsymbol{\tau}_t, \boldsymbol{y}_t)$. The PAC-Bayes framework for our proposed meta-optimization framework considers a Bayesian setting of DeepTime, having prior and posterior distributions of the meta and base parameters. The generalization bound is presented below, with the detailed proof in Appendix D.

Theorem 4.1. *(Generalization Bound) Let $\mathcal{Q}, Q$ be arbitrary distribution of ϕ, θ, which are defined in Equation (2) and Equation (3), and $\mathcal{P}, P$ be the prior distribution of ϕ, θ. Then for any $c_1, c_2 > 0$ and any $\delta \in (0,1]$, with probability at least $1-\delta$, the following inequality holds uniformly for all hyper-posterior distributions $\mathcal{Q}$,*

$$er(\mathcal{Q}) \leq \frac{c_1 c_2}{(1-e^{-c_1})(1-e^{-c_2})} \cdot \frac{1}{n}\sum_{k=1}^{n} \hat{er}(\mathcal{Q}, \mathcal{S}_k) + \frac{c_1}{1-e^{-c_1}} \cdot \frac{\mathrm{KL}(\mathcal{Q}||\mathcal{P}) + \log\frac{2}{\delta}}{nc_1} + \frac{c_1 c_2}{(1-e^{-c_2})(1-e^{-c_1})} \cdot \frac{\mathrm{KL}(Q||P) + \log\frac{2n}{\delta}}{(H+L)c_2} \tag{5}$$

where $er(\mathcal{Q})$ and $\hat{er}(\mathcal{Q}, \mathcal{S}_k)$ are the generalization error and training error of DeepTime, respectively.

Theorem 4.1 states that the expected generalization error of DeepTime is bounded by the empirical error plus two complexity terms. The first term represents the complexity between the meta distributions, $\mathcal{Q}, \mathcal{P}$, as well as between instances, converging to zero if we observe an infinitely long time-series ($n \to \infty$). The second term represents the complexity between the base distributions, Q, P, and of each instance, or equivalently, the lookback window and forecast horizon. This term converges to zero when there are a sufficient number of time steps in each instance ($H + L \to \infty$).

5. Experiments

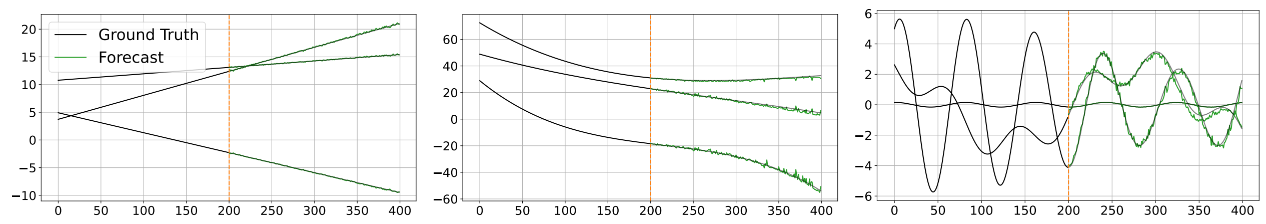
We evaluate DeepTime on both synthetic, and real-world datasets. We ask the following questions: (i) Is DeepTime, trained on a family of functions following the same parametric form, able to perform extrapolation on unseen functions? (ii) How does DeepTime compare to other forecasting models on real-world data? (iii) What are the key contributing factors to the good performance of DeepTime?

5.1. Experiments on Synthetic Data

We first consider DeepTime's ability to extrapolate on the following functions specified by some parametric form: (i) the family of linear functions, $y = ax + b$, (ii) the family of cubic functions, $y = ax^3 + bx^2 + cx + d$, and (iii) sums of sinusoids, $\sum_j A_j \sin(\omega_j x + \varphi_j)$. Parameters of the functions (i.e. $a, b, c, d, A_j, \omega_j, \varphi_j$) are sampled randomly (further details in Appendix E) to construct distinct tasks. A total of 400 time steps are sampled, with a lookback window length of 200 and forecast horizon of 200. Figure 4 demonstrates that DeepTime is able to perform extrapolation on unseen test functions/tasks after being trained via our meta-optimization formulation. It demonstrates an ability to approximate and adapt, based on the lookback window, linear and cubic polynomials, and even sums of sinusoids. Next, we evaluate DeepTime on real-world datasets, against state-of-the-art baselines.

5.2. Experiments on Real-world Data

Experiments are performed on 6 real-world datasets – Electricity Transformer Temperature (ETT), Electricity Consuming Load (ECL), Exchange, Traffic, Weather, and Influenza-like Illness (ILI) with full details in Appendix F. We evaluate the performance of our proposed approach using two metrics, the mean squared error (MSE) and mean absolute error (MAE) metrics. The datasets are split into train, validation, and test sets chronologically, following a 70/10/20 split for all datasets except for *ETTm2* which follows a 60/20/20 split, as per convention. The univariate benchmark selects the last index of the multivariate dataset as the target variable, following previous work (Xu et al., 2021). Preprocessing on the data is performed by standardization based on train set statistics. Hyperparameter selection is performed on only one value, the lookback length multiplier, $L = \mu * H$, which decides the length of the lookback window. We search through the values $\mu = [1, 3, 5, 7, 9]$, and select the best value based on the validation loss. Further implementation details on DeepTime are reported in Appendix G, and detailed hyperparameters are reported in Appendix H. Reported results for DeepTime are averaged over three runs, and standard deviation is reported in Appendix J.



(a) Linear (b) Cubic (c) Sum of Sinusoids

Figure 4. Predictions of DeepTime on three **unseen** functions for each function class. The orange line represents the split between lookback window and forecast horizon.

Table 1. Multivariate forecasting benchmark on long sequence time-series forecasting. Best results are highlighted in **bold**, and second best results are underlined.

Methods		DeepTime		NS Transformer		N-HiTS		ETSformer		FEDformer		Autoformer		Informer		LogTrans		GP	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.166**	0.257	0.192	0.274	0.176	**0.255**	0.189	0.280	0.203	0.287	0.255	0.339	0.365	0.453	0.768	0.642	0.442	0.422
	192	**0.225**	**0.302**	0.280	0.339	0.245	0.305	0.253	0.319	0.269	0.328	0.281	0.340	0.533	0.563	0.989	0.757	0.605	0.505
	336	**0.277**	**0.336**	0.334	0.361	0.295	0.346	0.314	0.357	0.325	0.366	0.339	0.372	1.363	0.887	1.334	0.872	0.731	0.569
	720	**0.383**	**0.409**	0.417	0.413	0.401	0.426	0.414	0.413	0.421	0.415	0.422	0.419	3.379	1.388	3.048	1.328	0.959	0.669
ECL	96	**0.137**	**0.238**	0.169	0.273	0.147	0.249	0.187	0.304	0.183	0.297	0.201	0.317	0.274	0.368	0.258	0.357	0.503	0.538
	192	**0.152**	**0.252**	0.182	0.286	0.167	0.269	0.199	0.315	0.195	0.308	0.222	0.334	0.296	0.386	0.266	0.368	0.505	0.543
	336	**0.166**	**0.268**	0.200	0.304	0.186	0.290	0.212	0.329	0.212	0.313	0.231	0.338	0.300	0.394	0.280	0.380	0.612	0.614
	720	**0.201**	**0.302**	0.222	0.321	0.243	0.340	0.233	0.345	0.231	0.343	0.254	0.361	0.373	0.439	0.283	0.376	0.652	0.635
Exchange	96	**0.081**	0.205	0.111	0.237	0.092	0.211	0.085	**0.204**	0.139	0.276	0.197	0.323	0.847	0.752	0.968	0.812	0.136	0.267
	192	**0.151**	**0.284**	0.219	0.335	0.208	0.322	0.182	0.303	0.256	0.369	0.300	0.369	1.204	0.895	1.040	0.851	0.229	0.348
	336	**0.314**	**0.412**	0.421	0.476	0.371	0.443	0.348	0.428	0.426	0.464	0.509	0.524	1.672	1.036	1.659	1.081	0.372	0.447
	720	**0.856**	**0.663**	1.092	0.769	0.888	0.723	1.025	0.774	1.090	0.800	1.447	0.941	2.478	1.310	1.941	1.127	1.135	0.810
Traffic	96	**0.390**	**0.275**	0.612	0.338	0.402	0.282	0.607	0.392	0.562	0.349	0.613	0.388	0.719	0.391	0.684	0.384	1.112	0.665
	192	**0.402**	**0.278**	0.613	0.340	0.420	0.297	0.621	0.399	0.562	0.346	0.616	0.382	0.696	0.379	0.685	0.390	1.133	0.671
	336	**0.415**	**0.288**	0.618	0.328	0.448	0.313	0.622	0.396	0.570	0.323	0.622	0.337	0.777	0.420	0.733	0.408	1.274	0.723
	720	**0.449**	**0.307**	0.653	0.355	0.539	0.353	0.632	0.396	0.596	0.368	0.660	0.408	0.864	0.472	0.717	0.396	1.280	0.719
Weather	96	0.166	0.221	0.173	0.223	**0.158**	**0.195**	0.197	0.281	0.217	0.296	0.266	0.336	0.300	0.384	0.458	0.490	0.395	0.356
	192	**0.207**	0.261	0.245	0.285	0.211	**0.247**	0.237	0.312	0.276	0.336	0.307	0.367	0.598	0.544	0.658	0.589	0.450	0.398
	336	**0.251**	**0.298**	0.321	0.338	0.274	0.300	0.298	0.353	0.339	0.380	0.359	0.359	0.578	0.523	0.797	0.652	0.508	0.440
	720	**0.301**	**0.338**	0.414	0.410	0.351	0.353	0.352	0.388	0.403	0.428	0.419	0.419	1.059	0.741	0.869	0.675	0.498	0.450
ILI	24	2.425	1.086	2.294	0.945	**1.862**	**0.869**	2.527	1.020	2.203	0.963	3.483	1.287	5.764	1.677	4.480	1.444	2.331	1.036
	36	2.231	1.008	**1.825**	**0.848**	2.071	0.969	2.615	1.007	2.272	0.976	3.103	1.148	4.755	1.467	4.799	1.467	2.167	1.002
	48	2.230	1.016	**2.010**	**0.900**	2.346	1.042	2.359	0.972	2.209	0.981	2.669	1.085	4.763	1.469	4.800	1.468	2.961	1.180
	60	**2.143**	0.985	2.178	**0.963**	2.560	1.073	2.487	1.016	2.545	1.061	2.770	1.125	5.264	1.564	5.278	1.560	3.108	1.214

Results We compare DeepTime to the following baselines for the multivariate setting, N-HiTS (Challu et al., 2022), ETSformer (Woo et al., 2022), Fedformer (Zhou et al., 2022) (we report the best score for each setting from the two variants they present), Autoformer (Xu et al., 2021), Informer (Zhou et al., 2021), LogTrans (Li et al., 2019), Non-stationary (NS) Transformer (Liu et al., 2022), and Gaussian Process (GP) (Rasmussen, 2003). For the univariate setting, we include additional univariate forecasting models, N-BEATS (Oreshkin et al., 2020), DeepAR (Salinas et al., 2020), Prophet (Taylor & Letham, 2018), and ARIMA. Baseline results are obtained from the respective papers. Table 1 and Table 7 (in Appendix I for space) summarizes the multivariate and univariate forecasting results respectively. DeepTime achieves state-of-the-art performance on 20 out of 24 settings in MSE, and 17 out of 24 settings in MAE on the multivariate benchmark, and also achieves competitive results on the univariate benchmark despite its simple architecture compared to the baselines comprising complex fully connected architectures and computationally intensive Transformer architectures.

5.3. Empirical Analysis and Ablation Studies

We first perform empirical analyses informed by the insights from our theoretical analysis. Theorem 4.1 states that generalization error is bounded by training error, and two complexity terms. Figure 5 and Table 2 analyse the test error (which approximates generalization error) as number of instances, n, and lookback window length, L, vary. For the first complexity term, controlled by denominator n, observed in Figure 5, test error decreases as n increases. For the second complexity term, controlled by the length

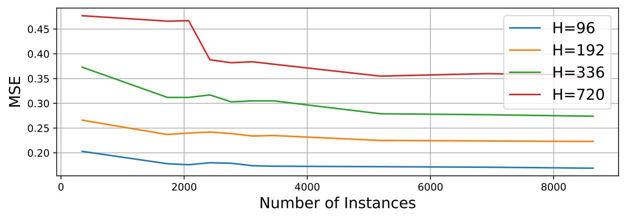


Figure 5. Analysis on the number of instances, n. MSE is measured as number of instances increases for multiple horizon lengths. Analysis is performed based on the ETTm2 dataset.

Table 2. Analysis on the lookback window length, based on a multiplier on horizon length, $L = \mu * H$. Results presented on the ETTm2 dataset. Best results are highlighted in **bold**.

Horizon	96		192		336		720	
μ	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
1	0.192	0.287	0.255	0.332	0.294	0.354	0.383	0.409
3	0.172	0.264	0.228	0.304	0.277	**0.336**	**0.371**	**0.403**
5	0.168	0.259	0.225	0.302	**0.275**	0.337	0.389	0.420
7	0.166	**0.257**	**0.223**	**0.300**	0.279	0.343	0.440	0.451
9	**0.165**	0.258	**0.223**	0.301	0.285	0.350	0.409	0.434

of lookback and horizon, since H is an experimental setting, we set perform a sensitivity analysis on the lookback length, setting $L = \mu * H$. Similarly, test error decreases as L increases, plateauing and even increasing slightly as L grows extremely large. We expect test error to plateau as the associated term goes to zero. As the number of instances available for training decreases as L grows large, the increase in test error can be attributed to a decrease in n.

In Table 3 we perform an ablation study on various backbone architectures, while retaining the differentiable closed-form ridge regressor. We observe a degradation when the random Fourier features layer is removed, due to the spectral bias problem which neural networks face (Rahaman et al., 2019; Tancik et al., 2020). DeepTime outperforms the SIREN variant of INRs which is consistent with observations INR literature. DeepTime also outperforms the RNN variant which is the model proposed in (Grazzi et al., 2021). This is a direct comparison between IMS historical-value models and time-index models, and highlights the benefits of a time-index models.

We perform an ablation study to understand how various training schemes and input features affect the performance of DeepTime. Table 4 presents these results. First, we observe that our meta-optimization formulation is a critical component to the success of DeepTime. We note that DeepTime without meta-optimization may not be a meaningful baseline since the model outputs are always the same regardless of the input lookback window. Including datetime features helps alleviate this issue, yet we observe that the inclusion of datetime features generally lead to a degradation in performance. In the case of DeepTime, we observed

Table 3. Ablation study on backbone models. DeepTime refers to our proposed approach, an INR with random Fourier features sampled from a range of scales. MLP refers to replacing the random Fourier features with a linear map from input dimension to hidden dimension. SIREN refers to an INR with periodic activations as proposed by Sitzmann et al. (2020b). RNN refers to an autoregressive recurrent neural network (inputs are the time-series values, $\boldsymbol{y}_t$). All approaches include differentiable closed-form ridge regressor. Further model details can be found in Appendix L.2.

Methods		DeepTime		MLP		SIREN		RNN	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.166**	**0.257**	0.186	0.284	0.236	0.325	0.233	0.324
	192	**0.225**	**0.302**	0.265	0.338	0.295	0.361	0.275	0.337
	336	**0.277**	**0.336**	0.316	0.372	0.327	0.386	0.344	0.383
	720	**0.383**	**0.409**	0.401	0.417	0.438	0.453	0.431	0.432

that the inclusion of datetime features lead to a much lower training loss, but degradation in test performance – this is a case of meta-learning memorization (Yin et al., 2020) due to the tasks becoming non-mutually exclusive (Rajendran et al., 2020). We also observe that the meta-optimization formulation is indeed superior to training a model from scratch for each lookback window. Finally, while we expect full MAML to always outperform the fast and efficient meta-optimization, in reality, there are many complications in such gradient-based bi-level optimization methods – they are difficult to optimize, and instable during training. Restricting the base parameters to only the last layer of the INR provides a useful prior which enables stable optimization and high generalization without facing these problems.

Additional sensitivity analysis on our proposed concatenated Fourier features, can be found in Appendix K, showing that it performs no worse than an extensive hyperparameter sweep on standard random Fourier features layer.

5.4. Computational Efficiency

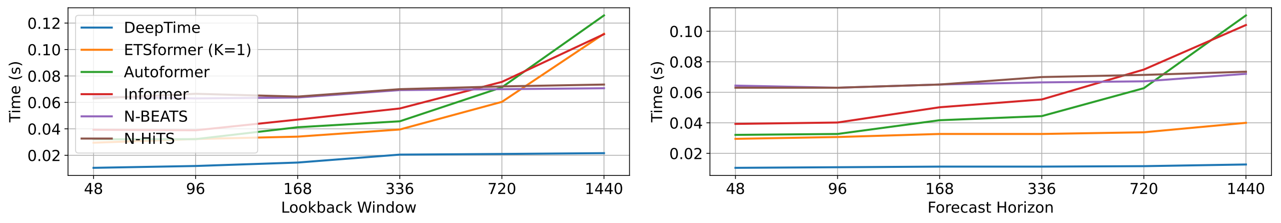
Finally, we analyse DeepTime's efficiency in both runtime and memory usage, with respect to both lookback window and forecast horizon lengths. The main bottleneck in computation for DeepTime is the matrix inversion operation in the ridge regressor, canonically of $\mathcal{O}(n^3)$ complexity. This is a major concern for DeepTime as n is linked to the length of the lookback window. As mentioned in (Bertinetto et al., 2019), the Woodbury formulation,

$$\boldsymbol{W}^* = \boldsymbol{Z}^T(\boldsymbol{Z}\boldsymbol{Z}^T + \lambda \boldsymbol{I})^{-1}\boldsymbol{Y}$$

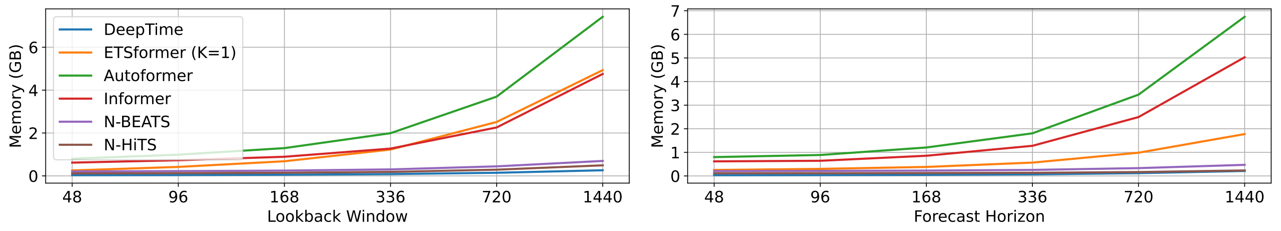
is used to alleviate the problem, leading to an $\mathcal{O}(d^3)$ complexity, where d is the hidden size hyperparameter, fixed to some value (see Appendix H). Figure 6 demonstrates that DeepTime is highly efficient, even when compared to efficient Transformer models, recently proposed for the long sequence time series forecasting task, as well as fully connected models.

Table 4. Ablation study on variants of DeepTime. Starting from the original version, we add (+) or remove (-) some component from DeepTime. *Datetime* refers to datetime features. *RR* stands for the differentiable closed-form **r**idge **r**egressor, removing it refers to replacing this module with a simple linear layer trained via gradient descent across all training samples (i.e. without meta-optimization formulation). *Local* refers to training an INR from scratch via gradient descent for each lookback window (RR is **not** used here, and there is no training phase). + *Finetune* refers to training an INR via gradient descent for each lookback window on top of having a training phase. *Full MAML* refers to performing gradient steps for the inner loop and backpropagating through them for the outer loop as in (Finn et al., 2017). Further details on the variants can be found in Appendix L.1.

Methods		DeepTime		+ Datetime		- RR		- RR + Datetime		+ Local		+ Local + Datetime		+ Finetune		+ Finetune + Datetime		Full MAML	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.166**	**0.257**	0.226	0.303	3.072	1.345	3.393	1.400	0.251	0.331	0.250	0.327	3.028	1.328	3.242	1.365	0.235	0.326
	192	**0.225**	**0.302**	0.309	0.362	3.064	1.343	3.269	1.381	0.322	0.371	0.323	0.366	3.043	1.341	3.385	1.391	0.295	0.361
	336	**0.277**	**0.336**	0.341	0.381	2.920	1.309	3.442	1.401	0.370	0.412	0.367	0.396	2.950	1.331	3.367	1.387	0.348	0.392
	720	**0.383**	**0.409**	0.453	0.447	2.773	1.273	3.400	1.399	0.443	0.449	0.455	0.461	2.721	1.253	3.476	1.407	0.491	0.484



(a) Runtime Analysis



(b) Memory Analysis

Figure 6. Computational efficiency benchmark on the ETTm2 multivariate dataset, on a batch size of 32. Runtime is measured for one iteration (forward + backward pass). Left: Runtime/Memory usage as lookback length varies, horizon is fixed to 48. Right: Runtime/Memory usage as horizon length varies, lookback length is fixed to 48. Further model details can be found in Appendix M.

6. Discussion

In this paper, we proposed DeepTime, a deep time-index model learned via a meta-optimization framework, to automatically learn a function form from time series data, rather than manually defining the representation function as per classical methods. DeepTime resolves issues arising for vanilla deep time-index models by splitting the learning process into inner and outer learning processes, where the outer learning process enables the deep time-index model to learn a strong inductive bias for extrapolation from data. We propose a fast and efficient instantiation of the meta-optimization framework, using a closed-form ridge regressor. We also enhance deep time-index models with a novel concatenated Fourier features module to efficiently learn high frequency patterns in time series. Our extensive empirical analysis shows that DeepTime, while being a much simpler model architecture compared to prevailing state-of-the-art methods, achieves competitive performance across forecasting benchmarks on real world datasets. We perform substantial ablation studies to identify the key components contributing to the success of DeepTime, and also show that it is highly efficient.

Limitations & Future Work Despite having verified DeepTime's effectiveness, we expect some underperformance in cases where the lookback window contains significant anomalies, or an abrupt change point. Next, while out of scope for our current work, a limitation that DeepTime faces is that it does not consider holidays and events. We leave the consideration of such features as a potential future direction, along with the incorporation of exogenous covariates and datetime features, whilst avoiding the incursion of the meta-learning memorization problem. Finally, time-index models are a natural fit for missing value imputation, as well as other time series intelligence tasks for irregular time series – this is another interesting future direction to extend deep time-index models towards.

References

Amit, R. and Meir, R. Meta-learning by adjusting priors based on extended pac-bayes theory. In *International Conference on Machine Learning*, pp. 205–214. PMLR, 2018.

Antoniou, A., Edwards, H., and Storkey, A. How to train your MAML. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJGven05Y7.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Basri, R., Galun, M., Geifman, A., Jacobs, D., Kasten, Y., and Kritchman, S. Frequency bias in neural networks for input of non-uniform density. In *International Conference on Machine Learning*, pp. 685–694. PMLR, 2020.

Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, B., Maddix, D., Turkmen, C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., et al. Neural forecasting: Introduction and literature overview. *arXiv preprint arXiv:2004.10240*, 2020.

Bertinetto, L., Henriques, J. F., Torr, P., and Vedaldi, A. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyxnZh0ct7.

Carbonneau, R., Laframboise, K., and Vahidov, R. Application of machine learning techniques for supply chain demand forecasting. *European Journal of Operational Research*, 184(3):1140–1154, 2008.

Catoni, O. Pac-bayesian supervised classification: the thermodynamics of statistical learning. *arXiv preprint arXiv:0712.0248*, 2007.

Challu, C., Olivares, K. G., Oreshkin, B. N., Garza, F., Mergenthaler, M., and Dubrawski, A. N-hits: Neural hierarchical interpolation for time series forecasting. *arXiv preprint arXiv:2201.12886*, 2022.

Chevillon, G. Direct multi-step estimation and forecasting. *Journal of Economic Surveys*, 21(4):746–785, 2007.

Corani, G., Benavoli, A., and Zaffalon, M. Time series forecasting with gaussian processes needs priors. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 103–117. Springer, 2021.

Dupont, E., Teh, Y. W., and Doucet, A. Generative models as distributions of functions. *arXiv preprint arXiv:2102.04776*, 2021.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Godfrey, L. B. and Gashler, M. S. Neural decomposition of time-series data for effective generalization. *IEEE transactions on neural networks and learning systems*, 29 (7):2973–2985, 2017.

Gong, M., Zhang, K., Schoelkopf, B., Tao, D., and Geiger, P. Discovering temporal causal relations from subsampled data. In *International Conference on Machine Learning*, pp. 1898–1906. PMLR, 2015.

Gong, M., Zhang, K., Schölkopf, B., Glymour, C., and Tao, D. Causal discovery from temporally aggregated time series. In *Uncertainty in artificial intelligence: proceedings of the... conference. Conference on Uncertainty in Artificial Intelligence*, volume 2017. NIH Public Access, 2017.

Grazzi, R., Flunkert, V., Salinas, D., Januschowski, T., Seeger, M., and Archambeau, C. Meta-forecasting by combining global deeprepresentations with local adaptation. *arXiv preprint arXiv:2111.03418*, 2021.

Harvey, A. C. and Shephard, N. 10 structural time series models. In *Econometrics*, volume 11 of *Handbook of Statistics*, pp. 261–302. Elsevier, 1993. doi: https://doi.org/10.1016/S0169-7161(05)80045-8. URL https://www.sciencedirect.com/science/article/pii/S0169716105800458.

Hyndman, R. J. and Athanasopoulos, G. *Forecasting: principles and practice*. OTexts, 2018.

Jeong, K.-J. and Shin, Y.-M. Time-series anomaly detection with implicit neural representation. *arXiv preprint arXiv:2201.11950*, 2022.

Kim, K.-j. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 95–104, 2018.

Laptev, N., Yosinski, J., Li, L. E., and Smyl, S. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, volume 34, pp. 1–5, 2017.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *ArXiv*, abs/1907.00235, 2019.

Liu, Y., Wu, H., Wang, J., and Long, M. Non-stationary transformers: Exploring the stationarity in time series forecasting. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=ucNDIDRNjjv.

Marcellino, M., Stock, J. H., and Watson, M. W. A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series. *Journal of econometrics*, 135(1-2):499–526, 2006.

McAllester, D. A. Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pp. 164–170, 1999.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Ord, K., Fildes, R. A., and Kourentzes, N. *Principles of business forecasting*. Wessex Press Publishing Co., 2017.

Oreshkin, B. N., Carpov, D., Chapados, N., and Bengio, Y. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1ecqn4YwB.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pp. 5301–5310. PMLR, 2019.

Rajendran, J., Irpan, A., and Jang, E. Meta-learning requires meta-augmentation. *Advances in Neural Information Processing Systems*, 33:5705–5715, 2020.

Rasmussen, C. E. Gaussian processes in machine learning. In *Summer school on machine learning*, pp. 63–71. Springer, 2003.

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. In *ICLR*, 2017.

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

Shalev-Shwartz, S. and Ben-David, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Sitzmann, V., Chan, E., Tucker, R., Snavely, N., and Wetzstein, G. Metasdf: Meta-learning signed distance functions. *Advances in Neural Information Processing Systems*, 33:10136–10147, 2020a.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020b.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Taieb, S. B., Hyndman, R. J., et al. *Recursive and direct multi-step forecasting: the best of both worlds*, volume 19. Citeseer, 2012.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33: 7537–7547, 2020.

Tancik, M., Mildenhall, B., Wang, T., Schmidt, D., Srinivasan, P. P., Barron, J. T., and Ng, R. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2846–2855, 2021.

Taylor, S. J. and Letham, B. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

Tewari, A., Thies, J., Mildenhall, B., Srinivasan, P., Tretschk, E., Wang, Y., Lassner, C., Sitzmann, V., Martin-Brualla, R., Lombardi, S., et al. Advances in neural rendering. *arXiv preprint arXiv:2111.05849*, 2021.

Woo, G., Liu, C., Sahoo, D., Kumar, A., and Hoi, S. Etsformer: Exponential smoothing transformers for time-series forecasting. *arXiv preprint arXiv:2202.01381*, 2022.

Xu, J., Wang, J., Long, M., et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34, 2021.

Yin, M., Tucker, G., Zhou, M., Levine, S., and Finn, C. Meta-learning without memorization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BklEFpEYwS.

Young, P. C., Pedregal, D. J., and Tych, W. Dynamic harmonic regression. *Journal of forecasting*, 18(6):369–394, 1999.

Yüce, G., Ortiz-Jiménez, G., Besbinar, B., and Frossard, P. A structured dictionary perspective on implicit neural representations. *arXiv preprint arXiv:2112.01917*, 2021.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of AAAI*, 2021.

Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., and Jin, R. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. *arXiv preprint arXiv:2201.12740*, 2022.

A. DeepTime Pseudocode

Algorithm 1 PyTorch-Style Pseudocode of Closed-Form Ridge Regressor

`mm`: matrix multiplication, `diagonal`: returns the diagonal elements of a matrix, `add_`: in-place addition
`linalg.solve` computes the solution of a square system of linear equations with a unique solution.

```
# X: inputs, shape: (n_samples, n_dim)
# Y: targets, shape: (n_samples, n_out)
# lambd: scalar value representing the regularization coefficient

n_samples, n_dim = X.shape

# add a bias term by concatenating an all-ones vector
ones = torch.ones(n_samples, 1)
X = cat([X, ones], dim=-1)

if n_samples >= n_dim:
    # standard formulation
    A = mm(X.T, X)
    A.diagonal().add_(softplus(lambd))
    B = mm(X.T, Y)
    weights = linalg.solve(A, B)
else:
    # Woodbury formulation
    A = mm(X, X.T)
    A.diagonal().add_(softplus(lambd))
    weights = mm(X.T, linalg.solve(A, Y))
w, b = weights[:-1], weights[-1:]
return w, b
```

Algorithm 2 PyTorch-Style Pseudocode of DeepTIMe

`rearrange`: einops style tensor operations
`mm`: matrix multiplication

```
# x: input time-series, shape: (lookback_len, multivariate_dim)
# lookback_len: scalar value representing the length of the lookback window
# horizon_len: scalar value representing the length of the forecast horizon
# inr: implicit neural representation

time_index = linspace(0, 1, lookback_len + horizon_len) # shape: (lookback_len + horizon_len)
time_index = rearrange(time_index, 't -> t 1') # shape: (lookback_len + horizon_len, 1)
time_reprs = inr(time_index) # shape: (lookback_len + horizon_len, hidden_dim)

lookback_reprs = time_reprs[:lookback_len]
horizon_reprs = time_reprs[-horizon_len:]
w, b = ridge_regressor(lookback_reprs, x)
# w.shape = (hidden_dim, multivariate_dim), b.shape = (1, multivariate_dim)
preds = mm(horizon_reprs, w) + b
return preds
```

B. Categorization of Forecasting Methods

Table 5. Categorization of time-series forecasting methods over the dimensions of time-index vs historical-value methods, and DMS vs IMS methods.

	Time-index	**Historical-value**
DMS	DeepTime Prophet Gaussian process Time-series regression	N-HiTS FEDformer ETSformer Autoformer Informer N-BEATS
IMS	-	DeepAR LogTrans ARIMA ETS

Multi-step Forecasts Forecasting over a horizon (multiple time steps) can be achieved via two strategies, direct multi-step, or iterative multi-step (Marcellino et al., 2006; Chevillon, 2007; Taieb et al., 2012), or even a mixture of both, but this has been less explored:

- **Direct Multi-step (DMS)**: A DMS forecaster directly predicts forecasts for the entire horizon. For example, to achieve a multi-step forecast of H time steps, a DMS forecaster simply outputs H values in a single forward pass.
- **Iterative Multi-step (IMS)**: An IMS forecaster iteratively predicts one step ahead, and consumes this forecast to make a subsequent prediction. This is performed iteratively, until the desired length is achieved.

C. Further Discussion on DeepTime as a Time-index Model

We first reiterate our definitions of time-index and historical-value models from Section 1. Time-index models are models whose predictions are *purely* functions of *current* time-index features. To perform forecasting (i.e. make predictions over some forecast horizon), time-index models make the predictions $\hat{\boldsymbol{y}}_{t+h} = f(\boldsymbol{\tau}_{t+h})$ for $h = 0, \dots, H-1$. Historical-value models predict the time-series value of future time step(s) as a function of past observations, and optionally, covariates.

Time-index Models	**Historical-value Models**
$\hat{\boldsymbol{y}}_t = f(\boldsymbol{\tau}_t)$	$\hat{\boldsymbol{y}}_{t+1} = f(\boldsymbol{y}_t, \boldsymbol{y}_{t-1}, \dots, \boldsymbol{z}_{t+1}, \boldsymbol{z}_t, \dots)$

Thus, forecasts are of the form, $\hat{\boldsymbol{y}}_{t+h} = \mathcal{A}(f, \boldsymbol{Y}_{t-L:t})(\boldsymbol{\tau}_{t+h})$, and as can be seen, while the inner loop optimization step is a function of past observations, the adapted time-index model it yields is purely a function of time-index features.

Next, we further discuss some subtleties of how time-index models interact with past observations. Some confusion regarding DeepTime's categorization as a time-index model may arise from the above simplified equation for predictions, since forecasts are now a function the lookback window due to the closed-form solution of $\boldsymbol{W}_t^{(K)*}$. In particular, that Equations (1) and (4) indicate that forecasts from DeepTime are in fact linear in the lookback window. However, we highlight that this is not in contradiction with our definition of historical-value and time-index models. Here, we differentiate between the *model*, $f \in \mathcal{H}$, and the *learning algorithm*, $\mathcal{A}$, which is specified in Equation (3) (the inner loop optimization). The learning algorithm $\mathcal{A} : \mathcal{H} \times \mathbb{R}^{L \times m} \to \mathcal{H}$ takes as input a model from the hypothesis class $\mathcal{H}$ and, past observations, returning a model minimizing the loss function $\mathcal{L}$. A time-index model is thus, still only a function of time-index features, while the learning algorithm is a function of past observations, i.e. $f, f_0 \in \mathcal{H}, f : \mathbb{R}^c \to \mathbb{R}^m, f = \mathcal{A}(f_0, \boldsymbol{Y}_{t-L:t})$. DeepTime as a forecaster, is a **deep time-index model endowed with a meta-optimization framework**. In order to perform forecasting, it has to perform an inner loop optimization defined by the learning algorithm, as highlighted in Equation (3). For the special case where we use the closed-form ridge regressor, the inner loop learning algorithm reduces to a form which is linear in the lookback window. Still, the deep time-index model is only a function of time-index features.

D. Generalization Bound for our Meta-optimization Framework

In this section, we derive a generalization bound for DeepTime under the PAC-Bayes framework (McAllester, 1999; Shalev-Shwartz & Ben-David, 2014). Our formulation follows Amit & Meir (2018) which introduces a meta-learning generalization bound. We assume that all instances share the same hypothesis space $\mathcal{H}$, sample space $\mathcal{Z}$ and loss function $\ell : \mathcal{H} \times \mathcal{Z} \to [0, 1]$. We observes n instances in the form of sample sets $\mathcal{S}_1, \ldots, \mathcal{S}_n$. The number of samples in each instance is $H + L$. Each instance $\mathcal{S}_k$ is assumed to be generated *i.i.d* from an unknown sample distribution $\mathcal{D}_k^{H+L}$. Each instance's sample distribution $\mathcal{D}_k$ is *i.i.d.* generated from an unknown meta distribution, E. Particularly, we have $\mathcal{S}_k = (z_{k-L}, \ldots, z_k, \ldots, z_{k+H-1})$, where $z_t = (\boldsymbol{\tau}_t, \boldsymbol{y}_t)$. Here, $\boldsymbol{\tau}_t$ is the time coordinate, and $\boldsymbol{y}_t$ is the time-series value. For any forecaster $h(\cdot)$ parameterized by θ, we define the loss function $\ell(h_\theta, z_t)$. We also define P as the prior distribution over $\mathcal{H}$ and Q as the posterior over $\mathcal{H}$ for each instance. In the meta-learning setting, we assume a hyper-prior $\mathcal{P}$, which is a prior distribution over priors, observes a sequence of training instances, and then outputs a distribution over priors, called hyper-posterior $\mathcal{Q}$. We restate Theorem 4.1 in the following:

Theorem D.1. *(Generalization Bound) Let $\mathcal{Q}, Q$ be arbitrary distribution of ϕ, θ, which are defined in Equation (2) and Equation (3), and $\mathcal{P}, P$ be the prior distribution of ϕ, θ. Then for any $c_1, c_2 > 0$ and any $\delta \in (0, 1]$, with probability at least $1 - \delta$, the following inequality holds uniformly for all hyper-posterior distributions $\mathcal{Q}$,*

$$\begin{aligned} er(\mathcal{Q}) \leq & \frac{c_1 c_2}{(1-e^{-c_1})(1-e^{-c_2})} \cdot \frac{1}{n} \sum_{k=1}^{n} \hat{er}(\mathcal{Q}, \mathcal{S}_k) \\ & + \frac{c_1}{1-e^{-c_1}} \cdot \frac{\mathrm{KL}(\mathcal{Q}||\mathcal{P}) + \log \frac{2}{\delta}}{n c_1} \\ & + \frac{c_1 c_2}{(1-e^{-c_2})(1-e^{-c_1})} \cdot \frac{\mathrm{KL}(Q||P) + \log \frac{2n}{\delta}}{(H+L) c_2} \end{aligned} \tag{6}$$

where $er(\mathcal{Q})$ and $\hat{er}(\mathcal{Q}, \mathcal{S}_k)$ are the generalization error and training error of DeepTime, respectively.

Proof. Our proof contains two steps. First, we bound the error within observed instances due to observing a limited number of samples. Then we bound the error on the instance environment level due to observing a finite number of instances. Both of the two steps utilize Catoni's classical PAC-Bayes bound (Catoni, 2007) to measure the error. Here, we give Catoni's classical PAC-Bayes bound.

Theorem D.2. *(Catoni's bound (Catoni, 2007)) Let $\mathcal{X}$ be a sample space, $P(X)$ a distribution over $\mathcal{X}$, Θ a hypothesis space. Given a loss function $\ell(\theta, X) : \Theta \times \mathcal{X} \to [0, 1]$ and a collection of M i.i.d random variables $(X_1, \ldots, X_M)$ sampled from $P(X)$. Let π be a prior distribution over hypothesis space. Then, for any $\delta \in (0, 1]$ and any real number $c > 0$, the following bound holds uniformly for all posterior distributions ρ over hypothesis space,*

$$P\left(\mathop{\mathbb{E}}_{X_i \sim P(X), \theta \sim \rho} [\ell(\theta, X_i)] \leq \frac{c}{1-e^{-c}} \left[\frac{1}{M} \sum_{m=1}^{M} \mathop{\mathbb{E}}_{\theta \sim \rho} [\ell(\theta, X_m)] + \frac{\mathrm{KL}(\rho||\pi) + \log \frac{1}{\delta}}{Mc} \right], \forall \rho \right) \geq 1 - \delta.$$

We first utilize Theorem D.2 to bound the generalization error in each of the observed instances. Let k be the index of instance, we have the definition of expected error and empirical error as follows,

$$er(\mathcal{Q}, \mathcal{D}_k) = \mathop{\mathbb{E}}_{P \sim \mathcal{Q}} \mathop{\mathbb{E}}_{h \sim Q(\mathcal{S}_k, P)} \mathop{\mathbb{E}}_{z \sim \mathcal{D}_k} \ell(h, z), \tag{7}$$

$$\hat{er}(\mathcal{Q}, \mathcal{S}_k) = \mathop{\mathbb{E}}_{P \sim \mathcal{Q}} \mathop{\mathbb{E}}_{h \sim Q(\mathcal{S}_k, P)} \frac{1}{H+L} \sum_{j=k-L}^{k+H-1} \ell(h, z_j). \tag{8}$$

Then, according to Theorem D.2, for any $\delta_k \sim (0, 1]$ and $c_2 > 0$, we have

$$P\left(er(\mathcal{Q}, \mathcal{D}_k) \leq \frac{c_2}{1-e^{-c_2}} \hat{er}(\mathcal{Q}, \mathcal{S}_k) + \frac{c_2}{1-e^{-c_2}} \cdot \frac{\mathrm{KL}(Q||P) + \log \frac{1}{\delta_k}}{(H+L) c_2} \right) \geq 1 - \delta_k. \tag{9}$$

Next, we bound the error due to observing a limited number of instances from the environment. Similarly, we have the definition of expected instance error as follows

$$\begin{aligned} er(\mathcal{Q}) &= \mathop{\mathbb{E}}_{D\sim E} \mathop{\mathbb{E}}_{\mathcal{S}\sim\mathcal{D}^{H+L}} \mathop{\mathbb{E}}_{P\sim\mathcal{Q}} \mathop{\mathbb{E}}_{h\sim Q(\mathcal{S},P)} \mathop{\mathbb{E}}_{z\sim D} \ell(h,z) \\ &= \mathop{\mathbb{E}}_{D\sim E} \mathop{\mathbb{E}}_{\mathcal{S}\sim\mathcal{D}^{H+L}} er(\mathcal{Q}, D). \end{aligned} \tag{10}$$

Then we have the definition of error across the n instances,

$$\frac{1}{n}\sum_{k=1}^{n} \mathop{\mathbb{E}}_{P\sim\mathcal{Q}} \mathop{\mathbb{E}}_{h\sim Q(\mathcal{S}_k,P)} \mathop{\mathbb{E}}_{z\sim\mathcal{D}_k} \ell(h,z) = \frac{1}{n}\sum_{k=1}^{n} er(\mathcal{Q},\mathcal{D}_k). \tag{11}$$

Then Theorem D.2 says that the following holds for any $\delta_0 \sim (0,1]$ and $c_1 > 0$, we have

$$P\left(er(\mathcal{Q}) \leq \frac{c_1}{1-e^{-c_1}} \frac{1}{n}\sum_{k=1}^{n} er(\mathcal{Q},\mathcal{D}_k) + \frac{c_1}{1-e^{-c_1}} \cdot \frac{\mathrm{KL}(\mathcal{Q}||\mathcal{P}) + \log\frac{1}{\delta_0}}{nc_1} \right) \geq 1-\delta_0. \tag{12}$$

Finally, by employing a union bound argument (Lemma 1, Amit & Meir (2018)), we could bound the probability of the intersection of the events in Equation (12) and Equation (9) For any $\delta > 0$, set $\delta_0 = \frac{\delta}{2}$ and $\delta_k = \frac{\delta}{2n}$ for $k = 1, \dots, n$,

$$\begin{aligned} P\Bigg(er(\mathcal{Q}) \leq \frac{c_1 c_2}{(1-e^{-c_1})(1-e^{-c_2})} \cdot \frac{1}{n}\sum_{k=1}^{n} \hat{er}(\mathcal{Q},\mathcal{S}_k) + \frac{c_1}{1-e^{-c_1}} \cdot \frac{\mathrm{KL}(\mathcal{Q}||\mathcal{P}) + \log\frac{2}{\delta}}{nc_1} \\ + \frac{c_1 c_2}{(1-e^{-c_2})(1-e^{-c_1})} \cdot \frac{\mathrm{KL}(Q||P) + \log\frac{2n}{\delta}}{(H+L)c_2} \Bigg) \geq 1-\delta. \end{aligned} \tag{13}$$

□

E. Synthetic Data

The training set for each synthetic data experiment consists 1000 functions/tasks, while the test set contains 100 functions/-tasks. We ensure that there is no overlap between the train and test sets.

Linear Samples are generated from the function $y = ax + b$ for $x \in [-1, 1]$. This means that each function/task consists of 400 evenly spaced points between -1 and 1. The parameters of each function/task (i.e. a, b) are sampled from a normal distribution with mean 0 and standard deviation of 50, i.e. $a, b \sim \mathcal{N}(0, 50^2)$.

Cubic Samples are generated from the function $y = ax^3 + bx^2 + cx + d$ for $x \in [-1, 1]$ for 400 points. Parameters of each task are sampled from a continuous uniform distribution with minimum value of -50 and maximum value of 50, i.e. $a, b, c, d \sim \mathcal{U}(-50, 50)$.

Sums of sinusoids Sinusoids come from a fixed set of frequencies, generated by sampling $\omega \sim \mathcal{U}(0, 12\pi)$. We fix the size of this set to be five, i.e. $\Omega = \{\omega_1, \dots, \omega_5\}$. Each function is then a sum of J sinusoids, where $J \in \{1, 2, 3, 4, 5\}$ is randomly assigned. The function is thus $y = \sum_{j=1}^{J} A_j \sin(\omega_{r_j} x + \varphi_j)$ for $x \in [0, 1]$, where the amplitude and phase shifts are freely chosen via $A_j \sim \mathcal{U}(0.1, 5), \varphi_j \sim \mathcal{U}(0, \pi)$, but the frequency is decided by $r_j \in \{1, 2, 3, 4, 5\}$ to randomly select a frequency from the set Ω.

The predictions from DeepTime in Figure 4 demonstrate some noise, likely stemming from the model's capability to learn high frequency features due to the use of implicit neural representations with random Fourier features. Since the synthetic data are all low frequency, smoothly changing functions, the noise is likely to be artifacts from the concatenated Fourier features layer, which should go away if the scale parameter of the Fourier features are carefully fine-tuned. However, the power of our proposed concatenated Fourier features layer is that the model is able to fit to both high and low frequency features without tuning, though at the expense of some noise as seen in the figure.

F. Datasets

ETT[1] (Zhou et al., 2021) - Electricity Transformer Temperature provides measurements from an electricity transformer such as load and oil temperature. We use the *ETTm2* subset, consisting measurements at a 15 minutes frequency.

ECL[2] - Electricity Consuming Load provides measurements of electricity consumption for 321 households from 2012 to 2014. The data was collected at the 15 mintue level, but is aggregated hourly.

Exchange[3] (Lai et al., 2018) - a collection of daily exchange rates with USD of eight countries (Australia, United Kingdom, Canada, Switzerland, China, Japan, New Zealand, and Singapore) from 1990 to 2016.

Traffic[4] - dataset from the California Department of Transportation providing the hourly road occupancy rates from 862 sensors in San Francisco Bay area freeways.

Weather[5] - provides measurements of 21 meteorological indicators such as air temperature, humidity, etc., every 10 minutes for the year of 2020 from the Weather Station of the Max Planck Biogeochemistry Institute in Jena, Germany.

ILI[6] - Influenza-like Illness measures the weekly ratio of patients seen with ILI and the total number of patients, obtained by the Centers for Disease Control and Prevention of the United States between 2002 and 2021.

[1] `https://github.com/zhouhaoyi/ETDataset`
[2] `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`
[3] `https://github.com/laiguokun/multivariate-time-series-data`
[4] `https://pems.dot.ca.gov/`
[5] `https://www.bgc-jena.mpg.de/wetter/`
[6] `https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html`

G. DeepTime Implementation Details

Optimization We train DeepTime with the Adam optimizer (Kingma & Ba, 2014) with a learning rate scheduler following a linear warm up and cosine annealing scheme. Gradient clipping by norm is applied. The ridge regressor regularization coefficient, λ, is trained with a different, higher learning rate than the rest of the meta parameters. We use early stopping based on the validation loss, with a fixed patience hyperparameter (number of epochs for which loss deteriorates before stopping). All experiments are performed on an Nvidia A100 GPU.

Model The ridge regression regularization coefficient is a learnable parameter constrained to positive values via a softplus function. We apply Dropout (Srivastava et al., 2014), then LayerNorm (Ba et al., 2016) after the ReLU activation function in each INR layer. The size of the random Fourier feature layer is set independently of the layer size, in which we define the total size of the random Fourier feature layer – the number of dimensions for each scale is divided equally.

H. DeepTime Hyperparameters

Table 6. Hyperparameters used in DeepTime.

	Hyperparameter	Value
Optimization	Epochs	50
	Learning rate	1e-3
	λ learning rate	1.0
	Warm up epochs	5
	Batch size	256
	Early stopping patience	7
	Max gradient norm	10.0
Model	Layers	5
	Layer size	256
	λ initialization	0.0
	Scales	$[0.01, 0.1, 1, 5, 10, 20, 50, 100]$
	Fourier features size	4096
	Dropout	0.1
	Lookback length multiplier, μ	$\mu \in \{1, 3, 5, 7, 9\}$

I. Univariate Forecasting Benchmark

Table 7. Univariate forecasting benchmark on long sequence time-series forecasting. Best results are highlighted in **bold**, and second best results are underlined.

Methods		DeepTime		N-HiTS		ETSformer		Fedformer		Autoformer		Informer		N-BEATS		DeepAR		Prophet		ARIMA		GP	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	0.065	0.186	0.066	**0.185**	0.080	0.212	**0.063**	0.189	0.065	0.189	0.088	0.225	0.082	0.219	0.099	0.237	0.287	0.456	0.211	0.362	0.125	0.273
	192	0.096	0.234	**0.087**	**0.223**	0.150	0.302	0.102	0.245	0.118	0.256	0.132	0.283	0.120	0.268	0.154	0.310	0.312	0.483	0.261	0.406	0.154	0.307
	336	0.138	0.285	**0.106**	**0.251**	0.175	0.334	0.130	0.279	0.154	0.305	0.180	0.336	0.226	0.370	0.277	0.428	0.331	0.474	0.317	0.448	0.189	0.338
	720	0.186	0.338	**0.157**	**0.312**	0.224	0.379	0.178	0.325	0.182	0.335	0.300	0.435	0.188	0.338	0.332	0.468	0.534	0.593	0.366	0.487	0.318	0.421
Exchange	96	**0.086**	0.226	0.093	**0.223**	0.099	0.230	0.131	0.284	0.241	0.299	0.591	0.615	0.156	0.299	0.417	0.515	0.828	0.762	0.112	0.245	0.165	0.311
	192	**0.173**	0.330	0.230	**0.313**	0.223	0.353	0.277	0.420	0.273	0.665	1.183	0.912	0.669	0.665	0.813	0.735	0.909	0.974	0.304	0.404	0.649	0.617
	336	0.539	0.575	**0.370**	**0.486**	0.421	0.497	0.426	0.511	0.508	0.605	1.367	0.984	0.611	0.605	1.331	0.962	1.304	0.988	0.736	0.598	0.596	0.592
	720	0.936	0.763	**0.728**	**0.569**	1.114	0.807	1.162	0.832	0.991	0.860	1.872	1.072	1.111	0.860	1.890	1.181	3.238	1.566	1.871	0.935	1.002	0.786

J. DeepTime Standard Deviation

Table 8. DeepTime main benchmark results with standard deviation. Experiments are performed over three runs.

(a) Multivariate benchmark.

Metrics		MSE (SD)	MAE (SD)
ETTm2	96	0.166 (0.000)	0.257 (0.001)
	192	0.225 (0.001)	0.302 (0.003)
	336	0.277 (0.002)	0.336 (0.002)
	720	0.383 (0.007)	0.409 (0.006)
ECL	96	0.137 (0.000)	0.238 (0.000)
	192	0.152 (0.000)	0.252 (0.000)
	336	0.166 (0.000)	0.268 (0.000)
	720	0.201 (0.000)	0.302 (0.000)
Exchange	96	0.081 (0.001)	0.205 (0.002)
	192	0.151 (0.002)	0.284 (0.003)
	336	0.314 (0.033)	0.412 (0.020)
	720	0.856 (0.202)	0.663 (0.082)
Traffic	96	0.390 (0.001)	0.275 (0.001)
	192	0.402 (0.000)	0.278 (0.000)
	336	0.415 (0.000)	0.288 (0.001)
	720	0.449 (0.000)	0.307 (0.000)
Weather	96	0.166 (0.001)	0.221 (0.002)
	192	0.207 (0.000)	0.261 (0.000)
	336	0.251 (0.000)	0.298 (0.001)
	720	0.301 (0.001)	0.338 (0.001)
ILI	24	2.425 (0.058)	1.086 (0.027)
	36	2.231 (0.087)	1.008 (0.011)
	48	2.230 (0.144)	1.016 (0.037)
	60	2.143 (0.032)	0.985 (0.016)

(b) Univariate benchmark.

Metrics		MSE (SD)	MAE (SD)
ETTm2	96	0.065 (0.000)	0.186 (0.000)
	192	0.096 (0.002)	0.234 (0.003)
	336	0.138 (0.001)	0.285 (0.001)
	720	0.186 (0.002)	0.338 (0.002)
Exchange	96	0.086 (0.000)	0.226 (0.000)
	192	0.173 (0.004)	0.330 (0.003)
	336	0.539 (0.066)	0.575 (0.027)
	720	0.936 (0.222)	0.763 (0.075)

K. Random Fourier Features Scale Hyperparameter Sensitivity Analysis

Table 9. Comparison of CFF against the optimal and pessimal scales as obtained from the hyperparameter sweep. We also calculate the change in performance between CFF and the optimal and pessimal scales, where a positive percentage refers to a CFF underperforming, and negative percentage refers to CFF outperforming, calculated as $\%$ change $= (\text{MSE}_{CFF} - \text{MSE}_{Scale})/\text{MSE}_{Scale}$.

		CFF		Optimal Scale (% change)		Pessimal Scale (% change)	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	0.166	0.257	0.164 (1.20%)	0.257 (-0.05%)	0.216 (-23.22%)	0.300 (-14.22%)
	192	0.225	0.302	0.220 (1.87%)	0.301 (0.25%)	0.275 (-18.36%)	0.340 (-11.25%)
	336	0.277	0.336	0.275 (0.70%)	0.336 (-0.22%)	0.340 (-18.68%)	0.375 (-10.57%)
	720	0.383	0.409	0.364 (5.29%)	0.392 (4.48%)	0.424 (-9.67%)	0.430 (-4.95%)

Table 10. Results from hyperparameter sweep on the scale hyperparameter. Best scores are highlighted in **bold**, and worst scores are highlighted in **bold red**.

Scale Hyperparam		0.01		0.1		1		5		10		20		50		100	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.216**	**0.300**	0.189	0.285	0.173	0.268	0.168	0.262	0.166	0.260	0.165	0.258	0.165	0.259	**0.164**	**0.257**
	192	**0.275**	**0.340**	0.264	0.333	0.239	0.317	0.225	**0.301**	0.225	0.303	0.224	0.302	0.224	0.304	**0.220**	0.301
	336	**0.340**	**0.375**	0.319	0.371	0.292	0.351	**0.275**	0.337	0.277	**0.336**	0.282	0.345	0.278	0.342	0.280	0.344
	720	**0.424**	0.430	0.405	0.420	0.381	0.412	**0.364**	**0.392**	0.375	0.408	0.410	**0.430**	0.396	0.423	0.406	0.429

We perform a comparison between the optimal and pessimal scale hyperparameter for the vanilla random Fourier features layer, against our proposed CFF. We first report the results on each scale hyperparameter for the vanilla random Fourier features layer in Table 10. As with the other ablation studies, the results reported in Table 10 is based on performing a hyperparameter sweep across lookback length multiplier, and selecting the optimal settings based on the validation set, and reporting the test set results. Then, the optimal and pessimal scales are simply the best and worst results based on Table 10. Table 9 shows that CFF achieves extremely low deviation from the optimal scale across all settings, yet retains the upside of avoiding this expensive hyperparameter tuning phase. We also observe that tuning the scale hyperparameter is extremely important, as CFF obtains up to a 23.22% improvement in MSE over the pessimal scale hyperparameter.

L. Ablation Studies Details

In this section, we list more details on the models compared to in the ablation studies section. Unless otherwise stated, we perform the same hyperparameter tuning for all models in the ablation studies, and use the same standard hyperparameters such as number of layers, layer size, etc.

L.1. Ablation study on variants of DeepTime

Datetime Features As each dataset comes with a timestamps for each observation, we are able to construct datetime features from these timestamps. We construct the following features:

1. Quarter-of-year
2. Month-of-year
3. Week-of-year
4. Day-of-year
5. Day-of-month
6. Day-of-week
7. Hour-of-day

8. Minute-of-hour

9. Second-of-minute

Each feature is initially an integer value, e.g. month-of-year can take on values in $\{0, 1, \ldots, 11\}$, which we subsequently normalize to a $[0, 1]$ range. Depending on the data sampling frequency, the appropriate features can be chosen. For the ETTm2 dataset, we used all features except second-of-minute since it is sampled at a 15 minute frequency.

RR Removing the ridge regressor module refers to replacing it with a simple linear layer, $\mathrm{Linear} : \mathbb{R}^d \to \mathbb{R}^m$, where $\mathrm{Linear}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{W} \in \mathbb{R}^{m \times d}$, $\boldsymbol{b} \in \mathbb{R}^m$. This corresponds to a straight forward INR, which is trained across all lookback-horizon pairs in the dataset.

Local For models marked "Local", we similarly remove the ridge regressor module and replace it with a linear layer. Yet, the model is not trained across all lookback-horizon pairs in the dataset. Instead, for each lookback-horizon pair in the validation/test set, we fit the model to the lookback window via gradient descent, and then perform prediction on the horizon to obtain the forecasts. A new model is trained from scratch for each lookback-horizon window. We perform tuning on an extra hyperparameter, the number of epochs to perform gradient descent, for which we search through $\{10, 20, 30, 40, 50\}$.

Finetune Models marked "Finetune" are similar to "Local", except that they have been trained on the training set first, and for each lookback-horizon pair in the test set, they are "finetuned" on the lookback window.

Full MAML "Full MAML" indicates the setting for which MAML is performed on the entire deep time-index model, by backpropagating through inner loop gradient steps as per Finn et al. (2017), rather than our proposed fast and efficient meta-optimization framework. Inner loop optimization is performed using the Adam optimizer, and is tuned over lookback length multiplier values of $\{1, 3, 5, 7, 9\}$, and inner loop iterations of $\{1, 5, 10\}$.

L.2. Ablation study on backbone models

For all models in this section, we retain the differentiable closed-form ridge regressor, to identify the effects of the backbone model used.

MLP The random Fourier features layer is a mapping from coordinate space to latent space $\gamma : \mathbb{R}^c \to \mathbb{R}^d$. To remove the effects of the random Fourier features layer, we simply replace it with a with a linear map, $\mathrm{Linear} : \mathbb{R}^c \to \mathbb{R}^d$.

SIREN We replace the random Fourier features backbone with the SIREN model which is introduced by (Sitzmann et al., 2020b). In this model, periodical activation functions are used, i.e. $\sin(\boldsymbol{x})$, along with specified weight initialization scheme.

RNN We use a 2 layer LSTM with hidden size of 256. Inputs are observations, $\boldsymbol{y}_t$, in an IMS fashion, predicting the next time step, $\boldsymbol{y}_{t+1}$.

M. Computational Efficiency Experiments Details

Trans/In/Auto/ETS-former We use a model with 2 encoder and 2 decoder layers with a hidden size of 512, as specified in their original papers.

N-BEATS We use an N-BEATS model with 3 stacks and 3 layers (relatively small compared to 30 stacks and 4 layers used in their orignal paper[7]), with a hidden size of 512. Note, N-BEATS is a univariate model and values presented here are multiplied by a factor of m to account for the multivariate data. Another dimension of comparison is the number of parameters used in the model. Demonstrated in Table 11, fully connected models like N-BEATS, their number of parameters scales linearly with lookback window and forecast horizon length, while for Transformer-based and DeepTime, the number of parameters remains constant.

[7]https://github.com/ElementAI/N-BEATS/blob/master/experiments/electricity/generic.gin

N-HiTS We use an N-HiTS model with hyperparameters as sugggested in their original paper (3 stacks, 1 block in each stack, 2 MLP layers, 512 hidden size). For the following hyperparameters which were not specified (subject to hyperparameter tuning), we set the pooling kernel size to $[2, 2, 2]$, and the number of stack coefficients to $[24, 12, 1]$. Similar to N-BEATS, N-HiTS is a univariate model, and values were multiplied by a factor of m to account for the multivariate data.

Table 11. Number of parameters in each model across various lookback window and forecast horizon lengths. The models were instantiated for the ETTm2 multivariate dataset (this affects the embedding and projection layers in Autoformer). Values for N-HiTS in this table are **not** multiplied by m since it is a global model (i.e. a single univariate model is used for all dimensions of the time-series).

Methods		Autoformer	N-HiTS	DeepTime
Lookback	48	10,535,943	927,942	1,314,561
	96	10,535,943	1,038,678	1,314,561
	168	10,535,943	1,204,782	1,314,561
	336	10,535,943	1,592,358	1,314,561
	720	10,535,943	2,478,246	1,314,561
	1440	10,535,943	4,139,286	1,314,561
	2880	10,535,943	7,461,366	1,314,561
	5760	10,535,943	14,105,526	1,314,561
Horizon	48	10,535,943	927,942	1,314,561
	96	10,535,943	955,644	1,314,561
	168	10,535,943	997,197	1,314,561
	336	10,535,943	1,094,154	1,314,561
	720	10,535,943	1,315,770	1,314,561
	1440	10,535,943	1,731,300	1,314,561
	2880	10,535,943	2,562,360	1,314,561
	5760	10,535,943	4,224,480	1,314,561